FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

OPC UA over Low Power Wireless Networks

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

Andreas Kirchberger
Matrikelnummer 0626507

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Thomas Frühwirth

Wien, 26. September 2018

Andreas Kirchberger	Wolfgang Kastner

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

OPC UA over Low Power Wireless Networks

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Technische Informatik

by

Andreas Kirchberger
Registration Number 0626507

to the Faculty of Informatics

at the TU Wien

Advisor: Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Thomas Frühwirth

Vienna, 26th September, 2018

Andreas Kirchberger   Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Andreas Kirchberger
Hermann Gmeiner-Gasse 3/25
3100 St.Pölten

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 26. September 2018

Andreas Kirchberger

Abstract

OPC Unified Architecture (OPC UA) is a common communication protocol in the industrial automation. It is standardized in IEC 62541 and enables the seamless communication of devices from different vendors. Classic OPC relied on the Windows Component Object Model (COM)/Distributed COM (DCOM) technology. In contrast to OPC, its successor OPC UA is platform independent and can be used in a variety of devices. This feature enables the use of OPC UA across all levels of the automation pyramid, even including the field level. The development of Low-Power Wireless Personal Area Network (LoWPAN) technologies made it possible to replace wired field bus networks with wireless standards, which are specifically designed for low power devices. To implement OPC UA on such devices without the need of gateways, IP based LoWPAN standards are needed.

The aim of this work is to show that OPC UA communication is possible over a LoWPAN network. Therefore, a software stack architecture combining OPC UA and IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) is defined and implemented based on open-source software. Furthermore, the implementation is compared to a setup using standard Wireless Local Area Network (WLAN) as communication protocol.

Kurzfassung

OPC Unified Architecture (OPC UA) ist ein gängiges Kommunikationsprotokoll in der industriellen Automatisierung. Es ist in der IEC 62541 genormt und ermöglicht die nahtlose Kommunikation von Geräten verschiedener Hersteller. Classic OPC setzt auf die Windows Component Object Model (COM)/Distributed COM (DCOM)-Technologie. Im Gegensatz zu OPC ist der Nachfolger OPC UA plattformunabhängig und kann in einer Vielzahl von Geräten eingesetzt werden. Diese Funktion ermöglicht den Einsatz von OPC UA auf allen Ebenen der Automatisierungspyramide, auch in der Feldebene. Die Entwicklung von Low-Power Wireless Personal Area Network (LoWPAN)-Technologien ermöglichte es, kabelgebundene Feldbusnetzwerke durch Funk-Standards zu ersetzen, die speziell für Geräte mit geringem Stromverbrauch entwickelt wurden. Um OPC UA auf solchen Geräten ohne Gateways zu implementieren, werden IP-basierte LoWPAN-Standards benötigt.

Ziel dieser Arbeit ist es zu zeigen, dass OPC UA Kommunikation über ein LoWPAN Netzwerk möglich ist. Daher wird eine Software-Stack-Architektur, die OPC UA und IPv6 über Low-Power Wireless Personal Area Network (6LoWPAN) kombiniert, definiert und auf Basis von Open-Source-Software implementiert. Darüber hinaus wird die Implementierung mit einem Setup unter Verwendung von Standard Wireless Local Area Network (WLAN) als Kommunikationsprotokoll verglichen.

Contents

CHAPTER 1

Introduction

1.1 Motivation and problem statement

OPC UA has become to an indispensable machine to machine communication protocol in modern industrial automation systems. The demand for control and data collection of every part of the automation systems is increasing. This is, for instance, due to the increased complexity, the endeavor towards Predictive Maintenance (PdM) and the use of data analytics in the Cloud. With the increasing number of sensors and controllers in such automation systems, the cabling effort and complexity is rapidly increasing. For this reason, LoWPAN can be interesting to reduce the cabling effort especially for sensors and control applications that are not time critical and have minor polling intervals. The mesh networking capability is a big advantage of LoWPAN over other wireless standards.

While LoWPAN protocols are not designed for high bandwidths and big Maximum Transmission Units (MTUs), OPC UA in contrast is primarily used on Ethernet networks with high bandwidths and powerful MTUs. It need to be prooved if OPC UA over LoWPAN is feasible and evaluated if the performance is practical for applications.

1.2 Methodological approach

The first task is to study the parts of the OPC UA specifications and the 6LoWPAN standard that are important for this work.

The second task is to select an OPC UA software stack that is appropriate for the solution and can be adapted to communicate with the selected Internet of Things (IoT) OS. Also a suitable Micro Controller Unit (MCU) and an IEEE 802.15.4 radio transceiver module that is supported by the OS needs to be selected. It is also important that the selected MCU has enough memory to fit the OS and the stack.

The third task is to setup the hardware and configure the OS to enable the radio transceiver module and setup the 6LoWPAN network. The OPC UA stack network implementation needs to be ported to the OS, this is the main challenge of the implementation.

The fourth and last task is to evaluate the comparison of the proposed solution against another solution regarding their response times and discuss the pros and cons of the found solution.

1.3 Structure of the work

The first Chapter 2 gives an introduction to parts of OPC UA, IEEE 802.15.4 and 6LoWPAN that are important for this work. Chapter 3 gives a survey of OPC UA stacks, IoT OS and LoWPAN networks. Chapter 4 shows the hardware setup and the implementation of the OPC UA stack on the OS. This section also presents the evaluation results of the found solution against another implementation.

CHAPTER 2

Technical background

2.1 OPC Unified Architecture

As the number of computer based control and monitor systems in the automation field continuously increased in the 1990s, a number of different bus systems and protocols arised to access the automation data. Most of them solved the same problem, but in a different and vendor-specific way. This made it difficult do build automation systems where parts from different vendors work together seamlessly.

Therefore, the vendors of Supervisory Control and Data Acquisition (SCADA) and Human Machine Interface (HMI) software, which had analogical challenges, united their efforts in a task force which was founded in 1995 by the companies Intuitive Technology, Intellution, Opto 22, Rockwell Software, and Fisher-Rosemount. Under the name OLE for Process Control (OPC) Foundation, they defined standards to unify the automation data access in Windows-based systems.

In August 1996, the initial specification for OPC Data Access was published. This first release was a success because of the reuse of COM and DCOM, which is a Microsoft Windows base technology and the focus on the main features. The focus on important features and the use of base Windows technologies allowed a quick adoption of the standard for the addressed use case. To meet the requirements of modern industrial applications, the OPC Foundation developed a variety of OPC specifications. Three of the most important specifications are:

OPC Data Access (DA) OPC DA defines the way to access process data in real-time.

OPC Alarm & Events (AE) With OPC AE it's possible to react to process events.

OPC Historical Data Access (HDA) OPC HDA was specified to get access to historical process data.

The use of Microsofts COM and DCOM technology in Classic OPC with its existing interprocess communication and network protocols led to fast developments and allowed a quick time-to-market. However, Classic OPC also has a number of disadvantages:

- Limitation to Microsoft systems
- Lack of a data model
- Deficient security features
- Not designed for Internet communication

Classic OPC found its way into many products in the automation pyramid, however the dependency on COM/DCOM and difficulties with remote data access disabled the use of OPC by manufactures in many other areas. With the goal to overcome all these issues without limited performance or features, OPC UA was developed. To fulfill the requirements in modern systems, OPC UA offers flexible and rich modeling instruments, as well as a system interface for a variety of platforms. To enable the use of OPC from embedded systems to SCADA and the automation pyramid up to Enterprise Resource Planning (ERP) systems (cf. Figure 2.1), scalability was also required.

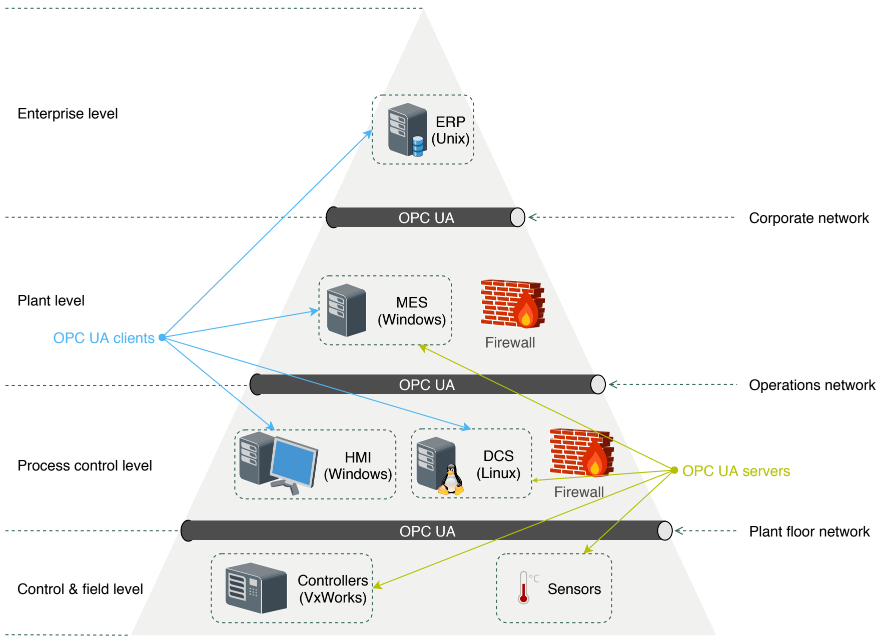

Figure 2.1: OPC UA within the automation pyramid

2.1.1 OPC UA Information Modeling

Classic OPC transfers only minimal information to understand the semantic of the transmitted data. For example the tag name and the engineering unit are transmitted with the provided data. With OPC UA it is possible to describe the semantic of the provided data. For example, a datapoint that measures the power consumption of a power circuit with Classic OPC would only deliver the measured power; with OPC UA meta information such as type of the sensor and the location of the sensor inside the power circuit can also be modeled. Figure 2.2 shows an example information model of a temperature sensor. For information modeling in OPC UA, these conventions are fundamental:

- An object-oriented way is applied with hierarchies and single inheritance
- Type information and instances shall be retrievable in the exact same manner
- Nodes can be related in different ways to connect information in a full mesh network, this provides the flexibility to depict different use cases
- To extend the functionality of OPC UA, type hierarchies and the type of references can be defined
- The implementation of the OPC UA information model is exclusively performed on the OPC Server.

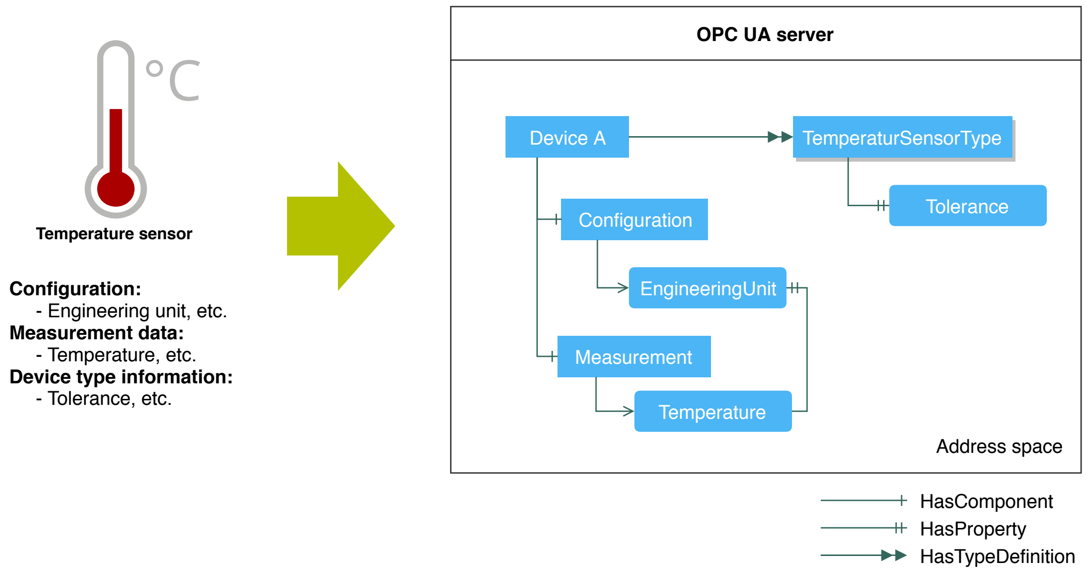

Figure 2.2: OPC UA information model example (adapted from [1])

2.1.2 OPC UA Transport and Encoding

The OPC UA standard describes two transport protocols that are used, UA Transmission Control Protocol (TCP) and Simple Object Access Protocol (SOAP)/Hypertext Transfer Protocol (HTTP). The connection from the UA client to the server is performed with these protocols on the network layer. Secure Channel and OPC UA Sessions are implemented on top of the network layer.

Service messages that are sent over the network layer can be encoded in different ways: OPC UA Binary, OPC UA XML or OPC UA JSON. Encoding describes the packing of the Service messages with their input and output parameters into a network format. OPC UA Binary provides fast encoding and decoding of data with minimal overhead. This is very important for devices with constraints regarding memory usage and processing power like embedded systems. This encoding is also preferred for applications where the throughput is very important because of the minimal overhead. OPC UA XML and OPC UA JSON are suitable for application and platforms where Extensible Markup Language (XML) or JavaScript Object Notation (JSON) are commonly used and already implemented. Both encodings are also straightforward to read and debug for human beings. Figure 2.3 shows the OPC UA transport profiles and the context of the different layers.

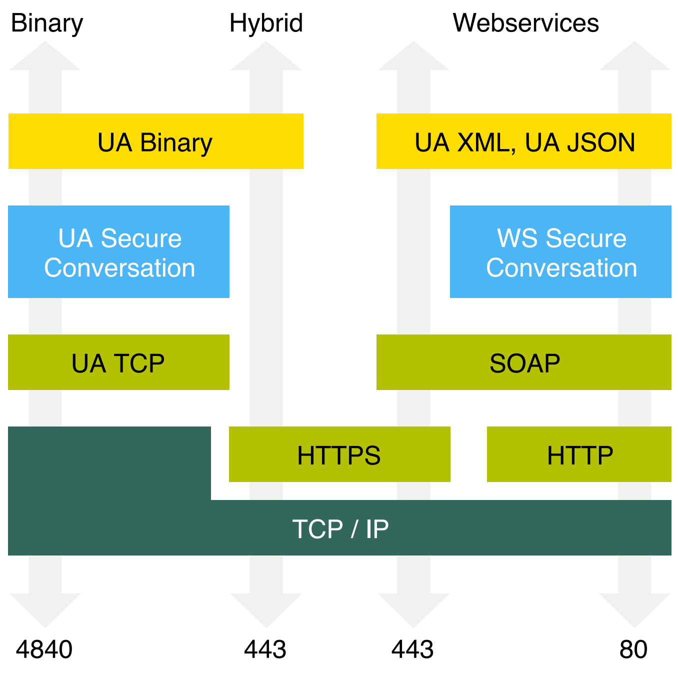

Figure 2.3: OPC UA transport profile (adapted from [2])

2.1.3 OPC UA Client Server interaction

The procedure of opening a connection to create a session for subsequent data requests is shown in Figure 2.4. In a client initiated connection, the client creates a *Transport Connection*, this is followed by a *Hello* request which shall be replied with an *Acknowledge* from the server. After the *Acknowledge* the client sends an *Open Secure Channel* request to the server. After receiving the *Open Secure Channel* response a session is created with a *Create Session* request and response. After this procedure *Data Requests* can be performed.

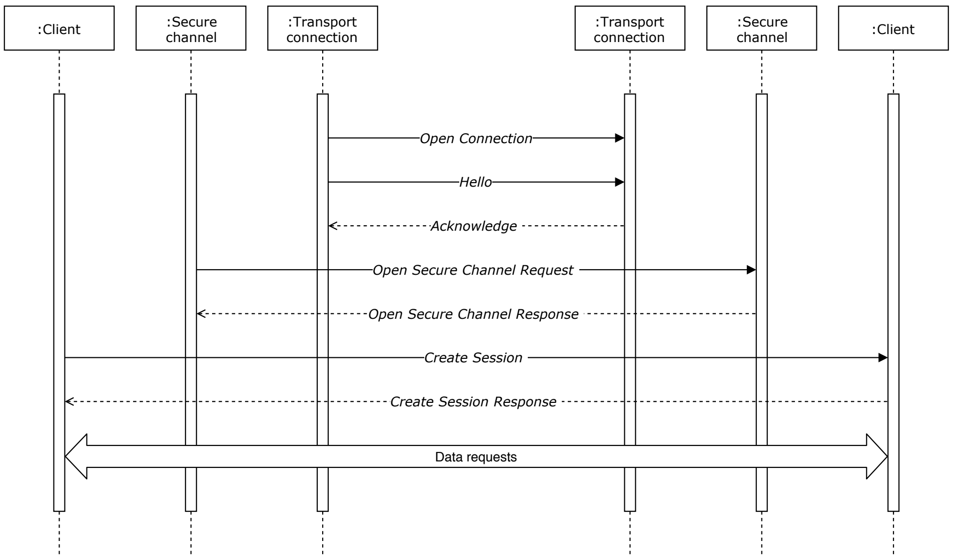

Figure 2.4: OPC UA connection establishment

2.2 IEEE 802.15.4

IEEE 802.15.4 [3] defines the two lowest layers (MAC, PHY) of the OSI model and leaves the upper layers to the developer. It was specially developed for low power and low cost applications.

2.2.1 Network topologies

IEEE 802.15.4 defines two topologies to meet the requirements for different applications: the star topology and the peer-to-peer topology. As seen in Figure 2.5, the star topology has a central point of communication called Personal Area Network (PAN) coordinator which is the primary controller of the PAN. Devices can only communicate via this point within the network. The second topology is the peer-to-peer topology, which allows direct communication of all devices with each other as long as they are in range of one another. This network topology also allows multi-hop routing, by which more complex network structures like mesh networks are enabled. A PAN coordinator is also implemented.

IEEE 802.15.4 specifies two different devices types:

- Full Function Devices (FFDs) have a complete set of Media Access Control (MAC) functionality and can operate as PAN coordinator and coordinator. FFDs are typically mains powered.
- Reduced Function Devices (RFDs) implement a reduced MAC functionality for minimal memory and resource usage. RFD devices are typically battery powered and sleep unless they need to send data. RFDs cannot communicate directly with each other but only via an intermediate FFD.

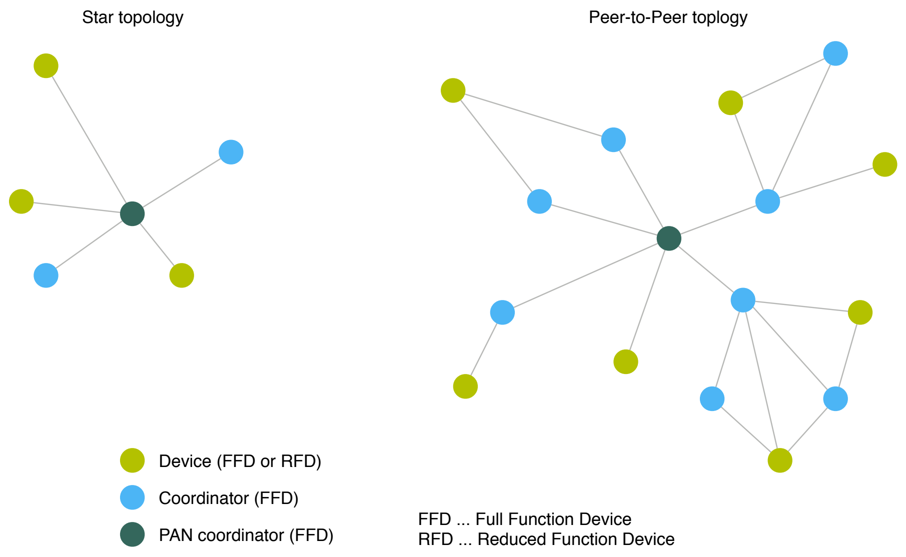

Figure 2.5: IEEE 802.15.4 star and peer-to-peer topology examples

2.3 6LoWPAN

LoWPAN is short for Low-power Wireless Personal Area Networks, these includes radio devices which are using the IEEE 802.15.4-2003 (cf. Section 2.2) standard which was defined by the Institute of Electrical and Electronics Engineers (IEEE). IEEE 802.15.4 was designed for low bandwidth connections between devices with very limited resources in computation, power and memory. Internet Protocol (IP) version 6 (IPv6), in contrast, was designed for high bandwidth Internet applications, which don't have significant constraints in all these resources. To bring IPv6 to Wireless IoT devices, 6LoWPAN was developed as an adaption layer between the link layer and the network layer oft the OSI reference model. The adaption layer has been developed by the Internet Engineering Task Force (IETF) and performs the compression of the IPv6 and higher layer headers. It also handles the fragmentation of the IPv6 packets as a result of the different MTU size (cf. Section 2.3.2).

2.3.1 IPv6

IPv6 was developed by the IETF and is the successor of the well known IP version 4 (IPv4). It was first defined in the Internet standard document RFC 2460 [4] and published in December 1998 as a new network layer (layer 3) of the Open Systems Interconnection (OSI) model. With the growing amount of devices that are connected to the Internet, it became evident that the number of IPv4 addresses ($\sim 4.3 * 10^9$) that are available would be exceeded in the near future. IPv6 uses a 128-bit address which defines $\sim 7.9 * 10^{28}$ times more addresses than IPv4 with its 32-bit address space. With a total amount of $\sim 3.4 * 10^{38}$ unique addresses this would result in $\sim 7 * 10^{23}$ IPv6 addresses per square meter evenly distributed over the whole earth, including the oceans. This should counteract the growing amount of IoT devices in the near future, where each device is supposed to have its own IP.

Address representation

The 128-bit IPv6 addresses are commonly represented as hexadecimal numerals in eight groups of 2 bytes. The eight groups are separated by colons (:). A typical IPv6 addresses would look like this:

```
2001:0DB8:0000:0000:0000:0000:0003:05B1
```

In comparison to IPv4, the IPv6 addresses are much longer, therefore the IETF defined RFC 5952 [5] to simplify and abbreviate the presentation of IPv6 addresses. The first method to reduce the length of an IPv6 address is to replace the longest sequence of consecutive zero bytes with a double colon (::). This is only allowed once. With this method, the address above could be reduced to:

```
2001:0DB8::0003:05B1
```

In addition, leading zeros may be suppressed in each 16-bit group. This would shorten the addresses above to:

```
2001:DB8::3:5B1
```

2.3.2 Packet Fragmentation

IPv6 requires a minimum packet size of 1280 bytes. The maximum packet size of an IEEE 802.15.4 frame is 127 bytes, this is why the IPv6 packets need to be fragmented into several link-level frames to satisfy the minimum MTU requirements of IPv6. The fragmentation headers (see Figure 2.6 and 2.7) are used if the payload datagram doesn't fit into a single IEEE 802.15.4 frame. The first fragment header (see Figure 2.6) consists of the `datagram_size` which defines the size of the entire IP packet and the `datagram_tag` which is a number that is equal in all link fragments of a payload datagram. All subsequent fragments (see Figure 2.7) have an additional `datagram_offset` field which defines the offset of the fragment inside the payload datagram, this offset is expressed in multiple of 8 bytes. That's why all link fragments apart from the last one need to be multiple of eight bytes long.

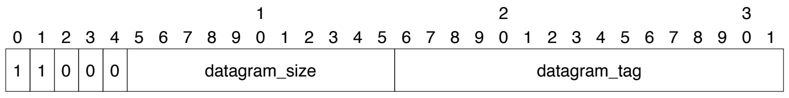

Figure 2.6: First IEEE 802.15.4 fragment header

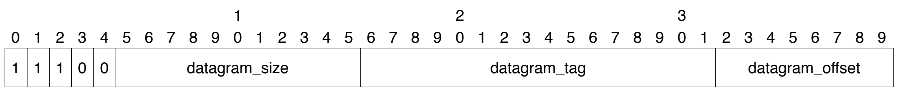

Figure 2.7: Subsequent IEEE 802.15.4 fragments header

2.3.3 6LoWPAN Header Compression

One of the tasks of 6LoWPAN is the compression of header information. It describes methods to reduce the amount of data in IPv6 and ensuring User Datagram Protocol (UDP) headers. This is important because of the small maximum MAC frame size of IEEE 802.15.4 which is 127 bytes, this allows a payload 102 bytes because 25 bytes a reserved for the MAC Header (MHR) and the MAC Footer (MFR). This amount of payload is reduced even further because the auxiliary security header is added by the link-layer to the MAC header. In the worst case, this would leave only 81 bytes for the IPv6 protocol. The size of an IPv6 header in an IPv6 frame is 40 bytes, thus, leaving only 41 bytes for higher layers. Encapsulated protocols at the transport layer like UDP with a 8 bytes header or TCP with a 20 bytes header would leave only a few bytes for the application-layer.

CHAPTER 3

OPC UA over Low Power Wireless Networks

3.1 Software stack architecture

The layers of the OPC UA stack do not correlate one-on-one with the OSI layer model. The stack is implemented in the layer 7 (application) of the OSI model and can be deployed on any layer 5, 6 or 7. To transmit OPC UA messages over LoWPAN the OPC UA stack is stacked on top of the TCP transport layer (cf. Figure 3.1).

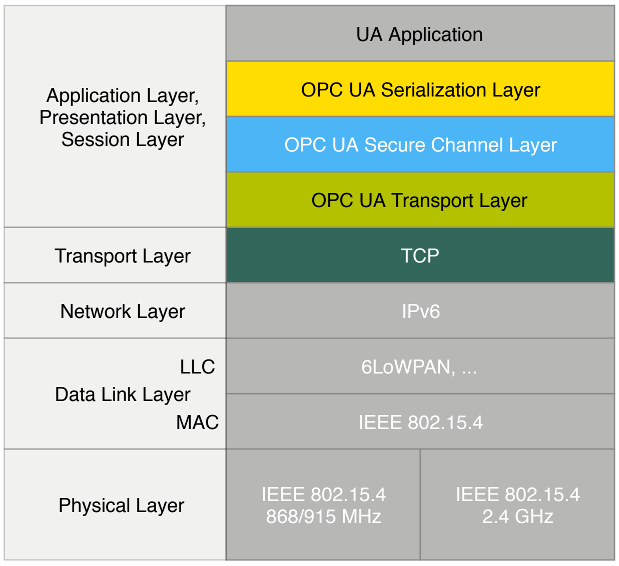

Figure 3.1: OPC UA stack and corresponding OSI layers

3.2 Software stack interaction

Figure 3.2 shows the interaction of the different software stacks. On the left side is the OPC UA server, on the right side the OPC UA client. The communication between the LoWPAN layers is simplified for presentation.

The OPC UA server first needs to prepare its TCP/IP layer by creating a socket, bind this socket to a port and register a callback to receive data requests.

An OPC UA client connects to a server by creating a socket and connecting to the given address and port. To establish this connection, a SYN message is sent to the server by the TCP/IP layer of the client. The server sends a SYN-ACK message to the client as acknowledgement. When this message is acknowledged by the client with an ACK message, the connection is established. To enable message fragmentation in TCP, the TCP receive window size can be defined to limit the amount of data that can be received by the client.

Following OPC UA messages are split into segments if their size is larger than the TCP window size. This is shown in the first loop of Figure 3.2. This loop is repeated until the full frame is transmitted. The OPC UA message fragments need to be defragmented by the OPC UA stack. The second loop acts equal to the first loop, OPC UA messages from the server to the client are fragmented and defragmented.

Figure 3.2 only illustrates how the TCP/IP connection is established and the first OPC UA Hello message, including the corresponding acknowledgement, is transmitted. Subsequent messages (c.f. Figure 2.4) are transmitted analogously.

3.3 Open source OPC UA stacks

A list of open source OPC UA stacks is given in Table 3.1. It shows the used programming language and the latest commit on Github.

	Language	**Latest commit**
open62541	C99	July 2018
node-opcua	Javascript	July 2018
FreeOpcUa	C++	January 2018
OPC Foundation UA Java	Java	June 2018
OPC Foundation UA ANSI C	ANSI C	July 2018

Table 3.1: OPC UA stacks

Open62541 is the first OPC UA stack in the Table 3.1. It's open source and royalty free. The name for this stack derives from the standard IEC 62541 that defines OPC UA. The source code is available on Github with a strong community of contributors.

Node-opcua is written in JavaScript and Node.js. Node.js makes it easy to develop web server applications, which makes node-opcua perfect for data visualization applications.

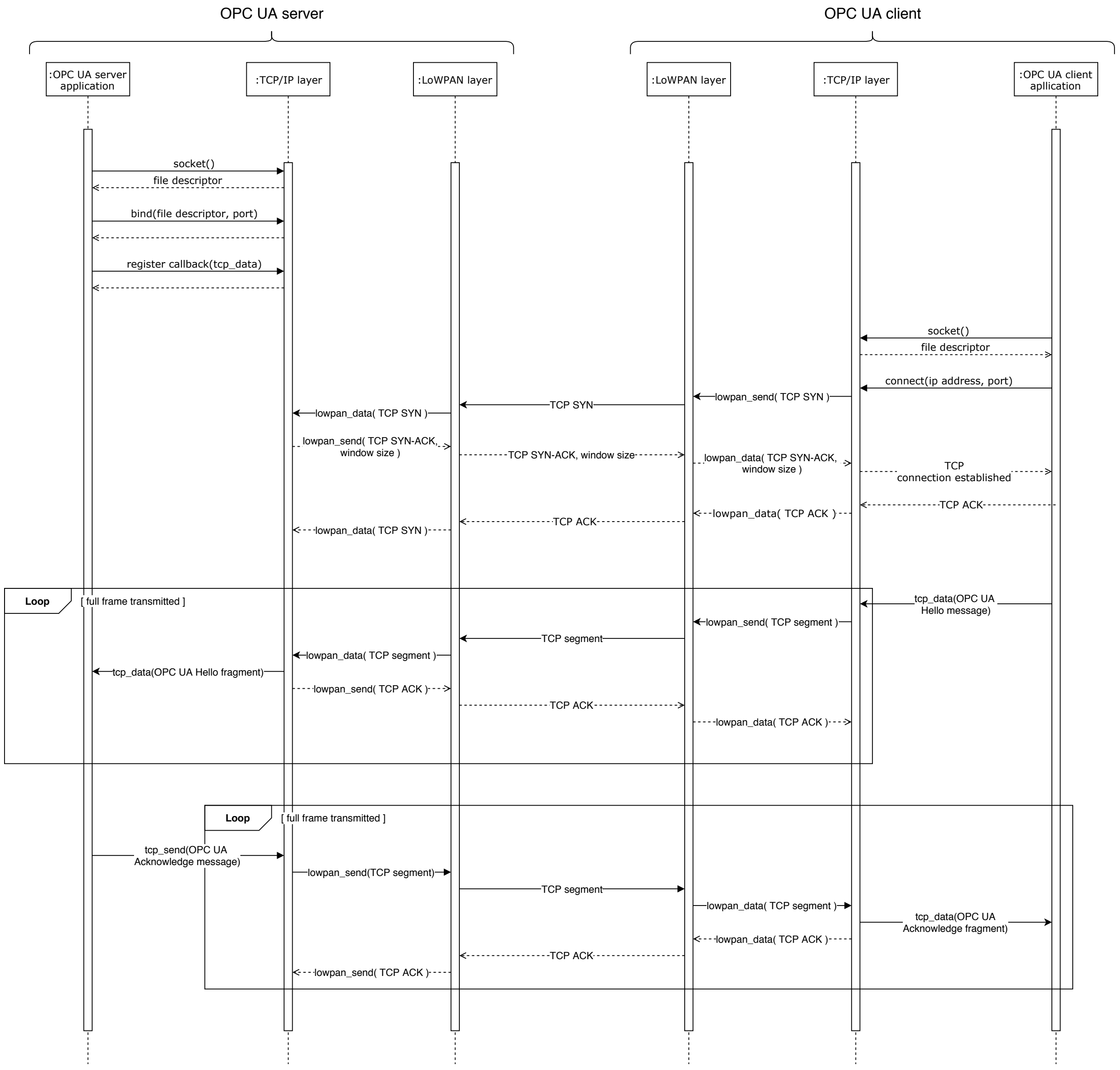
OPC UA server
OPC UA client
:OPC UA server application
:TCP/IP layer
:LoWPAN layer
:LoWPAN layer
:TCP/IP layer
:OPC UA client aplication
socket()
file descriptor
bind(file descriptor, port)
register callback(tcp_data)
socket()
file descriptor
connect(ip address, port)
lowpan_send(TCP SYN)
TCP SYN
lowpan_data(TCP SYN)
lowpan_send(TCP SYN-ACK, window size)
TCP SYN-ACK, window size
lowpan_data(TCP SYN-ACK, window size)
TCP connection established
TCP ACK
lowpan_data(TCP ACK)
TCP ACK
lowpan_data(TCP SYN)
Loop
[full frame transmitted]
tcp_data(OPC UA Hello message)
lowpan_send(TCP segment)
TCP segment
lowpan_data(TCP segment)
tcp_data(OPC UA Hello fragment)
lowpan_send(TCP ACK)
TCP ACK
lowpan_data(TCP ACK)
Loop
[full frame transmitted]
tcp_send(OPC UA Acknowledge message)
lowpan_send(TCP segment)
TCP segment
lowpan_data(TCP segment)
tcp_data(OPC UA Acknowledge fragment)
lowpan_data(TCP ACK)
TCP ACK
lowpan_send(TCP ACK)

Figure 3.2: Protocol stack interaction

FreeOpcUa is another open source OPC UA stack hosted on Github, it is developed in C++.

The OPC Foundation itself provides reference implementations in Java (**OPC Foundation UA Java**) and ANSI C (**OPC Foundation UA ANSI C**) which are also available on Github.

3.4 Operating systems for embedded devices

Table 3.2 shows a list of some OS for IoT devices, their support for C and C++, the supported network stacks and the latest commit on Github.

	C and C++ support	Network stack	Latest commit
Contiki	Partial C	uIP, RIME	March 2018
TinyOS	nesC	BLIP	May 2018
RIOT	C and C++	gnrc, OpenWSN, ccn-lite	July 2018

Table 3.2: Operating systems for embedded devices

Contiki is an OS which was developed for systems with memory-constrained resources like 8-bit MCUs. With the rise of more powerful MCUs, it became also available for 16-bit and 32-bit devices. For lightweight networking, micro IP (uIP) and Rime is available in Contiki. The uIP stack provides IPv4 and IPv6 networking support via uIPv6. Rime is also available in Contiki, which is a lightweight communication stack specially designed for LoWPAN.

TinyOS is an operating system for embedded platforms. It is specially designed for systems with memory-constrained resources. To meet these limitations, TinyOS was written in nesC. This programming language is a dialect of C and is optimized for low memory usage and bug prevention. For networking support in TinyOS, 6LoWPAN was implemented through Berkeley Low-power IP stack (BLIP).

In addition to Contiki and TinyOS, **RIOT** is another operating system for IoT applications. The kernel of RIOT ist written in C. For application development, C++ is also available. It supports a multitude of network stacks like gnrc, OpenWSN an ccn-lite. Gnrc is RIOTs default full 6LoWPAN network stack.

3.5 Low Power Wireless Networks

	Native IPv6	Link layer topology	Consortium
Bluetooth	Bluetooth Smart	Star	Bluetooth SIG
ZigBee	ZigBee IP	Mesh	ZigBee Alliance
6LoWPAN	yes	Mesh	IETF, Google

Table 3.3: LoWPAN protocols

Table 3.3 shows a selection of LoWPANs with native IPv6 support and their supported link layer topologies. Native IPv6 has been supported in **Bluetooth** [6] since version 4.1 with the specification of Bluetooth Smart and the Internet Standard RFC 7668 [7] defined by the IETF. The network topolgy is limited to star arrangement. With the specification of ZigBee IP by the ZigBee Alliance in 2012, native IPv6 with mesh networking is available in **ZigBee** [8]. **6LoWPAN** is defined in the Internet Standard RFC 4944 [9] and supports full mesh networking and native IPv6.

CHAPTER 4

Proof of concept

The open62541 stack was selected for the implementation of the OPC UA server in the test setup. This was due to the big open source community and the C99 programming language, which is compatible with most operating systems and compilers for embedded devices (cf. Table 3.2). First compilations of the open62541 stack showed that an MCU with sufficient amount of ram and flash was needed to fit the OPC UA stack and the operating system into the LoWPAN node. Therefore, the PIC32MX795F512L MCU was selected as a test platform. Contiki OS was selected as operating system because of the support for PIC32 MCUs. Finally, MRF24J40MB was selected as the IEEE 802.15.4 radio transceiver module as the driver for this chip is already available in Contiki OS.

4.1 Hardware setup

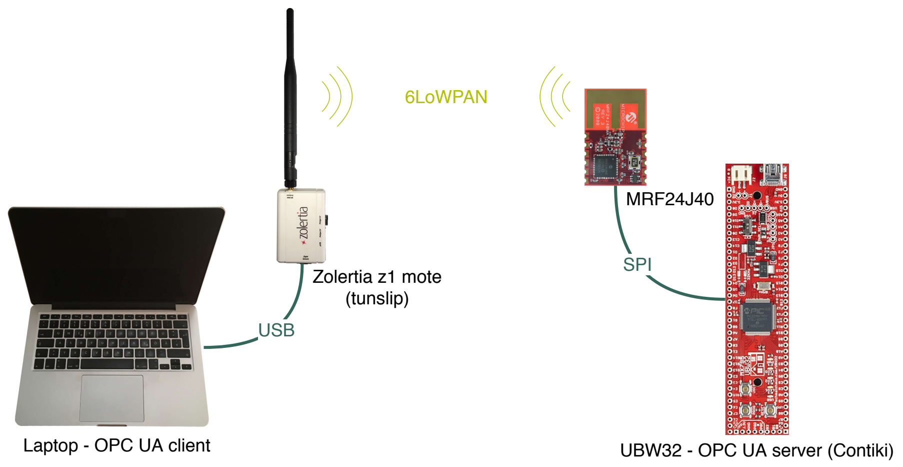

Figure 4.1: Hardware setup

Figure 4.1 shows the structure of the hardware setup and the chosen hardware components. On the left side of the 6LoWPAN radio channel, a laptop performs the role of the OPC UA client. The IPv6 network traffic is tunneled with tunslip over a serial IP tunnel to a Zolertia z1 mote over USB. The Zolertia z1 mote runs a border router firmware. On the right side, the UBW32 board with the PIC32MX795F512L MCU is used as an OPC UA server. The MRF24J40MB IEEE 802.15.4 radio transceiver is connected over Serial Peripheral Interface (SPI) with the PIC32.

4.2 Implementation

To implement open62541 in Contiki OS, some adaptions of the open62541 network implementation need to be done. Open62541 already has a network implementation, that can be found in the GitHub repository under `arch/ua_network_tcp.c`. Because of lack of Linux standard socket support in Contiki OS, the `arch/ua_network_tcp.c` file needs to be ported to the uIP socket Application Programming Interface (API), which is supported by Contiki OS.

For porting Contiki to a new hardware platform, the configuration settings need to be modified in the file `contiki-conf.h`. The following settings are important to enable 6LoWPAN, IEEE 802.15.4 and the MRF24J40MB radio driver:

```
#define NETSTACK_CONF_NETWORK     rime_driver
#define NETSTACK_CONF_FRAMER      framer_802154
#define NETSTACK_CONF_RADIO       mrf24j40_driver
```

Because TCP is a streaming protocol, open62541 needs to implement a function to merge fragmented messages and check for completeness. If the message is completed it is passed to the open62541 stack for processing. To enable packet fragmentation on the TCP level the `UIP_CONF_RECEIVE_WINDOW` is set to a value so that one TCP fragment fits into a single IEEE 802.15.4 frame. The size of the TCP receive window is the same as the TCP maximum segment size. This is defined in the `contiki-conf.h` under `UIP_CONF_TCP_MSS`. Following configuration settings need to be set:

```
#define UIP_CONF_RECEIVE_WINDOW   48
#define UIP_CONF_TCP_MSS          48
```

Another way to fit the OPC UA messages inside IEEE 802.15.4 frames would be to define the maximum message size which is returned by the OPC UA server in the *Hello - Acknowledge* message. This will notify the OPC UA client to send messages smaller than the maximum message size.

The process of packet reassembly is shown in the Figure 4.2. The uIP callback function *tcp_data_callback* is executed on every received TCP data frame. First the *msg_incomplete* variable, which is defaulted to false, is checked. It is true when a packet reassembly is in progress.

If the *msg_incomplete* variable is false the *opc_msg_data_len* is decoded from the first OPC UA fragment. Next the *input_data_len* is compared with the *opc_msg_data_len*. If the length is equal, *msg_incomplete* is set to false and *msg_incomplete_size* is set to 0. Because the OPC UA message fits into a single frame, the message can be directly passed as a *UA_WorkItem* to the open62541 stack. Otherwise the message is incomplete. Therefore the *msg_incomplete* variable is set to true and the *msg_incomplete_size* is incremented by the fragment length *input_data_len*. Now the received data fragment is appended to a *msg_incomplete_buffer*.

If the *msg_incomplete* variable is true, the message reassembly is in progress. The *msg_incomplete_size* is incremented by the fragment length *input_data_len* and the received data fragment is apppended to the *msg_incomplete_buffer*. If the *msg_incomplete_size* is greater or equal to the *opc_msg_data_len* the fragment is reassembled and send as a *UA_WorkItem* to the open62541 stack.

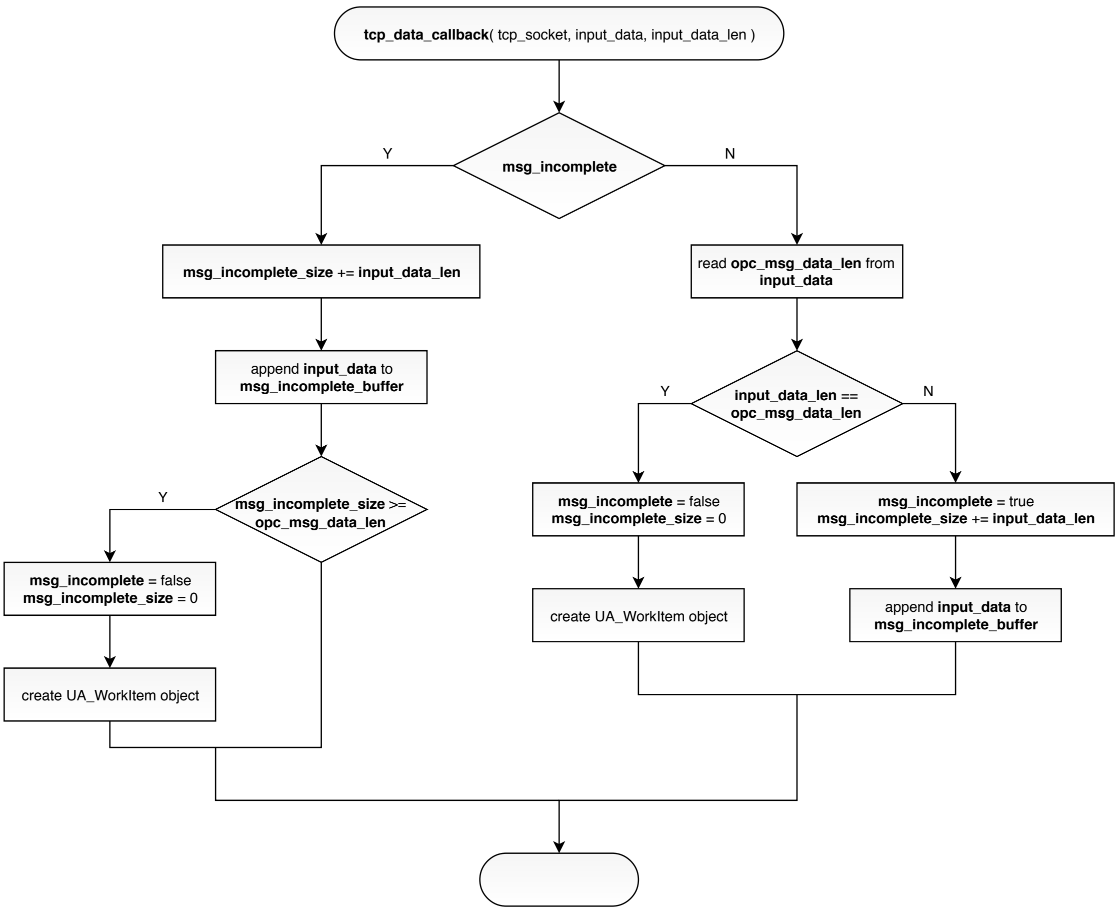
tcp_data_callback(tcp_socket, input_data, input_data_len)
msg_incomplete
Y
N
msg_incomplete_size += input_data_len
read opc_msg_data_len from input_data
append input_data to msg_incomplete_buffer
input_data_len == opc_msg_data_len
Y
N
msg_incomplete_size >= opc_msg_data_len
Y
msg_incomplete = false
msg_incomplete_size = 0
msg_incomplete = true
msg_incomplete_size += input_data_len
msg_incomplete = false
msg_incomplete_size = 0
create UA_WorkItem object
append input_data to msg_incomplete_buffer
create UA_WorkItem object

Figure 4.2: Flow chart of tcp_data and message reassembly

4.3 Evaluation

In this section the OPC UA over 6LoWPAN implementation is compared with an OPC UA over IEEE 802.11 WLAN [3] setup. The response time for data requests with different payload lengths simulated through a string data type is compared with each other.

4.3.1 Evaluation setup

The Figure 4.3 shows the two configurations for the evaluation. The laptop with the OPC UA client application remains the same for both measurements. In setup (a), 6LoWPAN and a PIC32 is used as an OPC UA server. In setup (b), a Raspberry Pi 3 is used as an WLAN hotspot and OPC UA server.

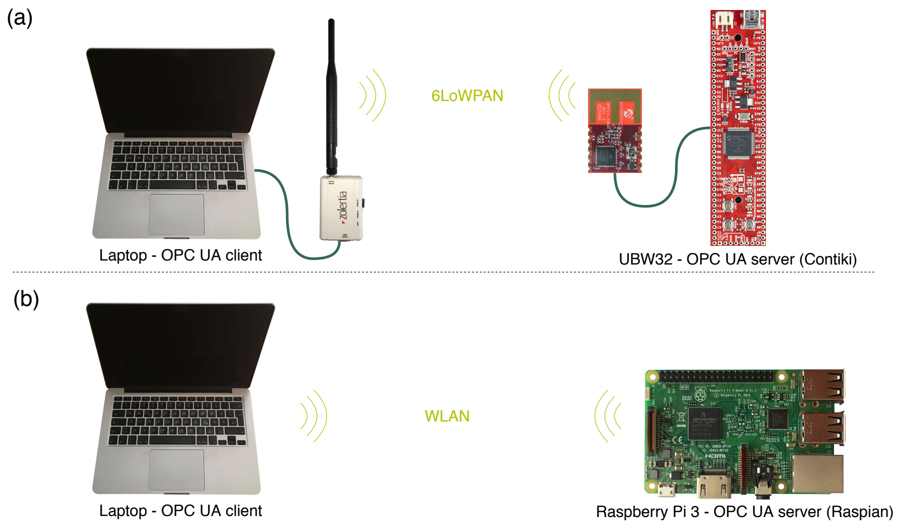
(a)
6LoWPAN
Laptop - OPC UA client
UBW32 - OPC UA server (Contiki)
(b)
WLAN
Laptop - OPC UA client
Raspberry Pi 3 - OPC UA server (Raspian)

Figure 4.3: Evaluation setups

4.3.2 Evaluation results

To get the evaluation results of setup (a) and (b), the OPC UA server on the PIC32 and the Raspberry Pi simulated a string variable with different lengths. The length of this variable, starting with 16 byte, was doubled with every measurement. To compare the two setups, the time from variable request to the response was measured. The variable was requested 100 times from the OPC UA server to get good results for the visualization. A median filter was applied to results to deal with very large values which would skew the final result. Such very high values could result for example from transmission errors.

Setup (a) in Figure 4.4 shows that for small messages the response time is, with about 600ms, already very high. The reason for this could be the low OPC UA message processing performance of the PIC32 and the fragmentation of OPC UA messages. The *OPC UA ReadRequest* message is in this test already 123 bytes long, with the selected window size of

48 bytes, this would need 3 TCP packets and 3 ACK to transmit the message. The OPC UA message with 16 byte payload also requires a 90 bytes *OPC UA ReadResponse* message in this case, which is fragmented into 2 TCP packets. This would need further investigation. This setup was tested and worked reliably for payloads less than 800 bytes, which is sufficient for the intended use case.

The second setup (b) in Figure 4.4 shows that the response time for payloads equal to or less 1024 bytes is smaller than 3ms. Even payloads as large as 262kB are transmitted much faster than the smallest payload in the evaluation setup (a).

The evaluation results showed that beside the transmission speed of the wireless technology, the MTU size plays an important role for a fast response time.

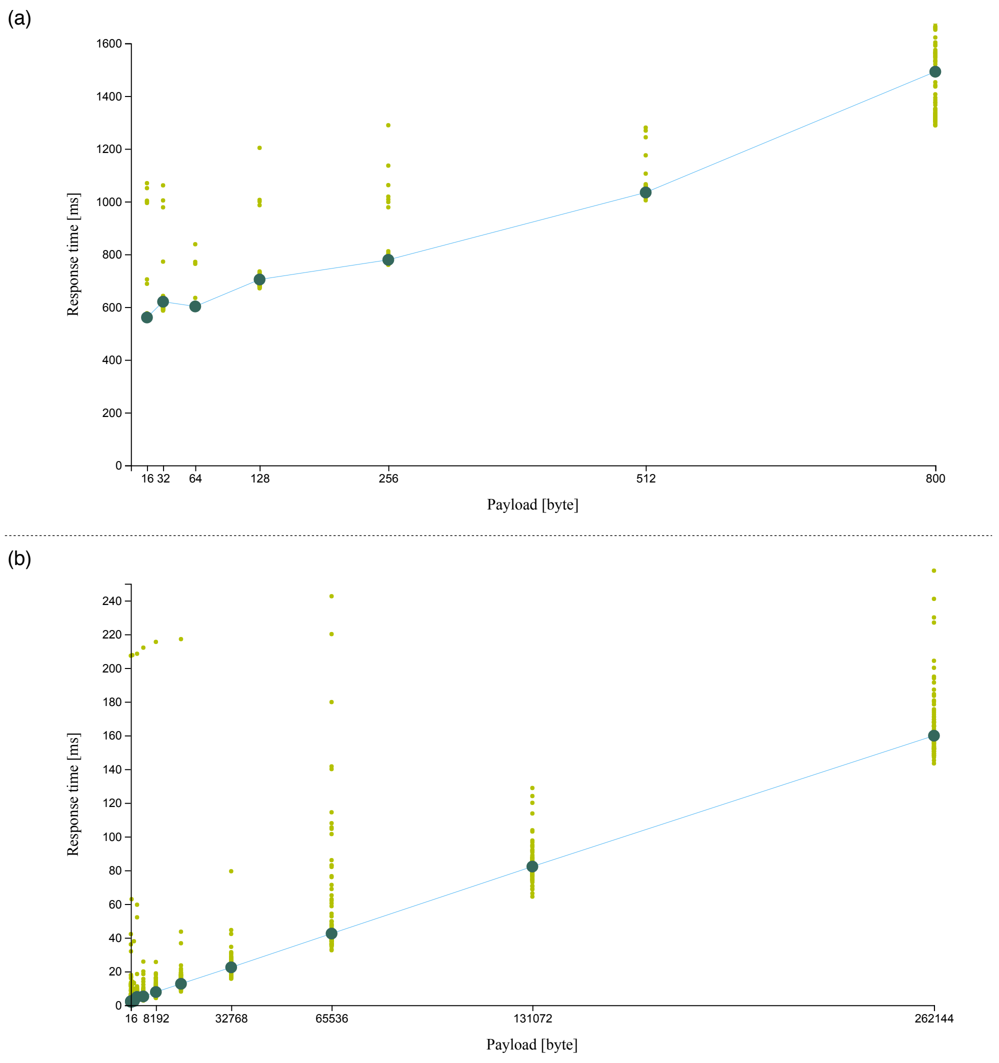

Figure 4.4: Measurement results

CHAPTER 5

Conclusion and outlook

In this work, the basics of *OPC Unified Architecture*, *IEEE 802.15.4* and *6LoWPAN* were treated. The theoretical idea of *OPC UA over Low Power Wireless Networks* was shown with the OPC UA stack on top of the LoWPAN stack. A survey of common OS for IoT devices and suitable LoWPAN protocols was conducted. The proof of concept showed that the implementation of OPC UA over LoWPAN is possible by using 6LoWPAN with Contiki OS and open62541 OPC UA stack on a PIC32 MCU. The evaluation of the response time measurements showed, that the response time for OPC UA messages over 6LoWPAN is in the range of 1s, which is high, but still feasible for time uncritical applications. The major problem are the small MTU size of 6LoWPAN, the lack of TCP header compression in 6LoWPAN, and the protocol overhead of OPC UA for basic messages. The evaluation also showed that OPC UA over WLAN is much faster, also with big payloads. WLAN is for many application much faster and more reliable, however compared to 6LoWPAN, WLAN doesn't support mesh networking and was not especially designed for low power usage. The proposed solution certainly has benefits for several applications.

For future work, the response time measurements for multi-hop LoWPAN networks may be interesting. This could determine if the proposed solution is suitable for applications, that require mesh networking capabilities. Another interesting development is the release of the new *OPC UA Part 14 - PubSub Specification* [10], which uses UDP on the transport layer and a publisher-subscriber model instead of the classic client-server model. The use of UDP instead of TCP leads to a reduced header overhead and enables the use of UDP/IPv6 header compression in 6LoWPAN. Another advantage of the publisher-subscriber model, in particular for IoT devices, is the reduced amount of traffic with the use of multicast UDP. The publishers and subscribers are loosely coupled, this would enable the devices to enter low-power sleep modes to reduce the power usage. Since open62541 supports this model since May 2018, this could be a topic for future work.

List of Figures

List of Tables

Acronyms

6LoWPAN IPv6 over Low-Power Wireless Personal Area Network. vii, 1, 2, 10, 14, 16, 18, 21

AE Alarm & Events. 3

API Application Programming Interface. 16

BLIP Berkeley Low-power IP stack. 14

COM Component Object Model. vii, 3, 4

DA Data Access. 3

DCOM Distributed COM. vii, 3, 4

ERP Enterprise Resource Planning. 4

FFD Full Function Device. 8

HDA Historical Data Access. 3

HMI Human Machine Interface. 3

HTTP Hypertext Transfer Protocol. 6

IEEE Institute of Electrical and Electronics Engineers. 8

IETF Internet Engineering Task Force. 8, 9, 14

IoT Internet of Things. 1, 2, 9, 14, 21

IP Internet Protocol. 8, 9, 25

IPv4 IP version 4. 9

IPv6 IP version 6. 8–10, 14, 16

JSON JavaScript Object Notation. 6

LoWPAN Low-Power Wireless Personal Area Network. vii, 1, 2, 11, 12, 14, 15, 21, 23

MAC Media Access Control. 8, 10, 26

MCU Micro Controller Unit. 1, 14–16, 21

MFR MAC Footer. 10

MHR MAC Header. 10

MTU Maximum Transmission Unit. 1, 8, 9

OPC OLE for Process Control. 3–5

OPC UA OPC Unified Architecture. vii, 1, 4–7, 11–13, 15–19, 21, 23

OSI Open Systems Interconnection. 9

PAN Personal Area Network. 7, 8

PdM Predictive Maintenance. 1

RFD Reduced Function Device. 8

SCADA Supervisory Control and Data Acquisition. 3, 4

SOAP Simple Object Access Protocol. 6

SPI Serial Peripheral Interface. 16

TCP Transmission Control Protocol. 6, 10, 16

UDP User Datagram Protocol. 10, 21

uIP micro IP. 14, 16

WLAN Wireless Local Area Network. vii, 18, 21

XML Extensible Markup Language. 6

Bibliography

[1] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC Unified Architecture*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[2] OPC Foundation. OPC Unified Architecture - Wegbereiter der 4. Industriellen (R)Evolution. `https://www.iosb.fraunhofer.de/servlet/is/21752/OPC-UA-Wegbereiter-der-I40.pdf?command=downloadContent&filename=OPC-UA-Wegbereiter-der-I40.pdf`, 2013. [Online; accessed 24-July-2018].

[3] IEEE Computer Society. Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans). *IEEE Std 802.15.4-2006*, 2006.

[4] Robert M. Hinden and Dr. Steve E. Deering. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, December 1998.

[5] Seiichi Kawamura and Masanobu Kawashima. A Recommendation for IPv6 Address Text Representation. RFC 5952, August 2010.

[6] SIG Bluetooth. Bluetooth core specification version 4.1. *Specification of the Bluetooth System*, 2013.

[7] Johanna Nieminen, Teemu Savolainen, Markus Isomaki, Basavaraj Patil, Zach Shelby, and Carles Gomez. IPv6 over BLUETOOTH(R) Low Energy. RFC 7668, October 2015.

[8] IEEE Computer Society LAN MAN Standards Committee et al. Wireless lan medium access control (mac) and physical layer (phy) specifications. *ANSI/IEEE Std. 802.11-1999*, 1999.

[9] Gabriel Montenegro, Jonathan Hui, David Culler, and Nandakishore Kushalnagar. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, September 2007.

[10] OPC Foundation. *OPC Unified Architecture Specification - Part 14: PubSub*, Release 1.04 edition, 2018.